Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Transformers as Generic Deep Neural Networks: Exact Compilation, the Cost of Removing Activation–Activation Products, and Multi-Agent Attention Clustering

Sergey Salishev

s.salischev@spbu.ru

St Petersburg University

Zeev Volkovich

Braude College of Engineering Karmiel

Oleg Granichin

St Petersburg University

Article

Keywords: Transformer compilation, neural networks, attention, quantization, neural processing units, multi-agent systems

Posted Date: November 17th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-11491535/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Transformers as Generic Deep Neural Networks: Exact Compilation, the Cost of Removing Activation–Activation Products, and Multi-Agent Attention Clustering

Sergey Salishev[1,*], Zeev Volkovich[2], and Oleg Granichin[1]

[1]St. Petersburg State University, Saint Petersburg 199034, Russia
[2]Department of Software Engineering, Braude College of Engineering, Karmiel 2161002, Israel
[*]Corresponding author: s.salischev@spbu.ru

Abstract

Transformers are deployed as a special model class. Neural processing units (NPU) add dedicated operators for attention and normalization, and quantization relies on Transformer-specific algorithms. We show the computation does not force this. Every Transformer block, with attention, LayerNorm and a GELU MLP, compiles exactly into a generic deep neural network built from fixed affine maps, routing and the square, sigmoid and inverse square root, plus GELU or its sigmoid approximation. Without activation–activation products, every transform-wise compilation adds at least N^2 scalar nonlinear evaluations per head for dense scores. Keeping that one operation type removes this cost; its elementwise form, present on every surveyed NPU, suffices with the sigmoid and inverse square root. For top-k attention with exchangeable tail scores, we prove that exact or conditional-mean tail mass gives an unbiased, minimum-mean-square-error estimate of the attention distribution, while tail drop is conditionally biased. When queries or agents cluster, centroids choose the keys first, reducing the number of computed scores from N^2 to Nk per head. On DeiT-Tiny, MambaIRv2-Light and Pythia-70M the compiled models match their references with all keys, standard post-training quantization applies, and the exact tail-mass estimate beats tail dropping with half as many keys on DeiT-Tiny and, for $k \leq 8$, on Pythia-70M.

Keywords: Transformer compilation; neural networks; attention; quantization; neural processing units; multi-agent systems

Introduction

Transformers [1] are treated as a model class apart from the deep neural networks (DNNs) that accelerators and tools were first built for.

First, hardware. Accelerators built for convolutional networks run a fixed operator set and hand everything else to the host. The Edge TPU compiler cuts the model at the first unsupported operation and runs the rest on the CPU [2]. NVIDIA's Jetson DLA has no matrix-multiply layer [3]. On phones, llm.npu runs the linear layers on the NPU and keeps LayerNorm and attention on the CPU or GPU [4]. Vendors answer with Transformer-specific operators and units for attention, softmax and normalization [5–7].

Second, quantization. Post-training quantization (PTQ) of Transformers relies on dedicated algorithms. Liu et al. search quantization scales with a ranking-aware objective [8], and PTQ4ViT adds twin-uniform quantizers for the post-softmax and post-GELU activations [9]. FQ-ViT adds power-of-two factors for LayerNorm and a log-integer softmax [10]. RepQ-ViT reparameterizes quantization scales [11], IGQ-ViT groups channels per instance [12], I-BERT replaces the nonlinearities by integer polynomials [13], and SmoothQuant moves activation outliers into the weights of language models [14].

The computation does not force this split.

Exact compilation. Every Transformer block, with LayerNorm, scaled dot-product attention, softmax and a GELU MLP, compiles exactly into a generic DNN of fixed affine maps, scalar nonlinearities and routing, with no product of two activations (Theorem 1). We call it the strict graph. Squares replace products, through $ab = \frac{1}{2}[(a+b)^2 - a^2 - b^2]$ where the squares of the factors are shared and through polarization otherwise, and no exponential or logarithm is needed: squares, the sigmoid and the inverse square root suffice, with GELU or its approximation $x\,\sigma(1.702x)$ as the MLP activation.

The cost of strictness. Since the strict graph has no dedicated layers, standard DNN tools apply to it. But removing products of two activations has a price. For free arguments, any compilation without them adds at least N^2 scalar nonlinear evaluations per head at $QK^\top$ and Nd at PV, whatever the scalar functions, depth or width (Theorem 2).

One more operation type. Adding activation–activation products removes the price. Their elementwise form, the Hadamard product, is on every NPU we survey, and it suffices: with it, a block needs only the sigmoid and the inverse square root besides its MLP activation, and only softmax adds work, $2N^2$ evaluations per head (Proposition 1). With the exponential and the reciprocal it adds no pointwise work (Proposition 2), and with matrix multiplication of two activations the extended compilation adds no work of either kind (Theorem 3). On a GPU, for $k \leq 64$, the Hadamard and strict graphs run at 1.0 to 1.3 and 1.2 to 1.5 times the latency of this extended graph. Theorems 2 and 3 make this operation type necessary and sufficient for compilation without added pointwise work. Practice already uses this operation set. ONNX defines LayerNormalization and Softmax as functions over it [15, 16], current NPUs expose it [6, 17, 18], and convolutional networks with squeeze-and-excitation gating already use the elementwise product [19, 20].

Clustered attention. The attention weights of a query cluster on a few keys [21–23] when its scores have a gap (Lemma 1). Queries cluster too [24, 25], and queries close to one point share their top keys (Lemma 2). Top-k attention keeps the k largest scores of each query [21, 22], the topological rule of bird flocks [26]. We add an estimate of the omitted probability mass, spread uniformly over the dropped keys. For exchangeable tail scores, exact or conditional-mean tail mass gives the unbiased, minimum-mean-square-error attention-distribution estimate; tail drop is conditionally biased (Theorem 4). Centroids of the query clusters [25] choose the keys before the scores, so the strict graph forms only Nk scores per head (Corollary 2, Proposition 5). One step of the Motsch–Tadmor flocking model is one attention head (Proposition 6).

We test the compilation on three model families: DeiT-Tiny [27] for image classification on Oxford-IIIT Pet [28] and ImageNet-1k [29], MambaIRv2-Light [30] for super-resolution, and Pythia-70M [31] for language modeling.

Related work. Universal approximation gives some network within each error on a compact set [32, 33]. Forming products from squares is classical: tables of quarter-squares multiply by

polarization [34], Winograd's inner product forms the self-terms of each row once and halves the products of a matrix multiplication [35], and expressivity proofs for ReLU networks use $ab = \frac{1}{2}[(a+b)^2 - a^2 - b^2]$ [36]. Compilers such as Glow map a model onto the operators a device provides [37]. Privacy-preserving inference decomposes attention, softmax and LayerNorm into primitive operations for cryptographic protocols, usually with polynomial approximations [38–40]. Multiplication-free training replaces every product by an approximate piecewise-affine operation and trains on that arithmetic [41]. Training-time alternatives simplify LayerNorm [42] or remove it [43, 44]. Top-k attention saves memory on long contexts [21–23], usually by renormalizing the kept scores. Scatterbrain adds a low-rank estimate of the non-selected attention to a sparse one [45], Combiner attends to the rest of the sequence through region summaries [46], and MagicPIG replaces top-k selection by sampling to avoid its bias [47]. Clustered attention chooses the kept keys by query centroids and gives the other keys the weights of the centroid [25]; we use its key selection with calibrated centroids. Lookup tables, piecewise-linear units and integer approximations implement scalar nonlinearities in hardware [7, 48, 49].

What is new. Unlike universal approximation, the compiled graph is exact on all inputs, reuses the trained weights and has a known size. Unlike compilers and privacy-preserving decompositions, we ask which small operator set expresses a whole block exactly. New are the lower bound that prices product-free compilation and the extended construction that removes the price. Unlike multiplication-free training, we compile a trained model, keep LayerNorm, and remove only products of two activations. Our tail-mass estimate is simpler than those of Scatterbrain, Combiner and clustered attention. It is a fixed affine map for a constant mass. For clustered queries, new are the cost bounds for the selected scores and the flocking step as one attention head.

Results

Theory

All proofs are in Methods.

A generic DNN vocabulary

We describe a DNN by the types of operations its nodes perform. Three types cover plain convolutional networks and multilayer perceptrons.

Definition 1 (Strict basis)**.** Fix a set Φ of scalar functions. A graph over the strict basis $\mathcal{B}_{\text{strict}}(\Phi)$ is a finite acyclic graph whose nodes are of three types:

1. *weight–activation contraction*: a fixed affine map $x \mapsto Ax + b$, where A and b are parameters of the graph and do not depend on the input;
2. *unary map*: $x \mapsto \phi(x)$ applied entrywise, with $\phi \in \Phi$;
3. *routing*: copy, reshape, permutation, concatenation, broadcast, masking, and selection of entries by comparisons of computed values (max, TopK, gather), with a fixed tie-breaking rule.

No node multiplies two quantities that both depend on the input; a square is a unary map. For every Φ, we call $\mathcal{B}_{\text{strict}}(\Phi)$ a *product-free* basis and its graphs product-free graphs.

Convolutions, fully connected layers, batch normalization at inference, residual sums and average pooling are fixed affine maps; max pooling is routing. ReLU, GELU and other activations are unary maps. Gating, as in squeeze-and-excitation blocks [19], multiplies two activations and is not in this basis. The set Φ is a parameter. Shifts such as $x + \varepsilon$ and scalings such as $x/2$ belong to the preceding affine node, so a unary set needs only the shape of each function.

Definition 2 (Strict unary set)**.** $\Phi_{\text{strict}} = \{(\cdot)^2,\ \sigma,\ (\cdot)^{-1/2},\ \text{GELU}\}$, with $\sigma(t) = 1/(1 + e^{-t})$.

Each function has one role. The square forms second moments and products: $ab = \frac{1}{2}[(a+b)^2 - a^2 - b^2]$ where the squares of the factors are shared across products, as for the rows of Q, K and V and in LayerNorm, and polarization, $ab = \frac{1}{4}[(a+b)^2 - (a-b)^2]$, for an isolated product. The sigmoid gives the exponential, $e^t = \sigma(t)/\sigma(-t)$. The inverse square root normalizes LayerNorm and, squared, gives the reciprocal, $1/x = (x^{-1/2})^2$ for $x > 0$. GELU is the MLP activation (Definition 3). The set has no exponential and no logarithm.

Definition 3 (Transform sets)**.** The Transformer transform set is

$$\mathcal{T}_{\text{Tr}} = \{\text{Aff},\ \text{LN},\ \text{Score},\ \text{softmax},\ \text{Mix},\ \text{MLP}_{\text{GELU}}\},$$

where

$$\text{Aff}(x) = Ax + b, \qquad \text{LN}(x) = \beta + \gamma \odot \frac{x - \text{mean}(x)}{\sqrt{\text{Var}(x) + \varepsilon}},$$

$$\text{Score}(Q, K) = QK^\top/\sqrt{d}, \qquad \text{Mix}(P, V) = PV,$$

$$\text{MLP}_{\text{GELU}}(x) = A_2\text{GELU}(A_1x + b_1) + b_2, \qquad \text{GELU}(x) = \tfrac{x}{2}\big(1 + \text{erf}(x/\sqrt{2})\big),$$

and softmax acts on each row of scores. A model may use the approximation $\text{GELU}(x) \approx x\,\sigma(1.702x)$ [50] instead, which changes the model. Then GELU is no longer a basic operation: it is omitted from every unary set, and the sigmoid is added where missing. Here $\text{Var}(x) = \text{mean}_i[(x_i - \text{mean}(x))^2]$, $\varepsilon > 0$ and $A, b, \beta, \gamma, A_1, b_1, A_2, b_2$ are fixed parameters, and one attention head has N tokens, head dimension d, $Q, K, V \in \mathbb{R}^{N\times d}$ and $P \in \mathbb{R}^{N\times N}$.

A compiled graph serves a fixed context capacity N, and the construction gives one graph for each N. Shorter inputs are padded and masked, and causal models use the causal mask. Masks are fixed routing and affine maps.

Every Transformer block compiles exactly

Theorem 1 (Exact compilation into the strict basis)**.** *Every finite Transformer block composed of transforms in $\mathcal{T}_{\text{Tr}}$ is computed exactly by a finite graph over $\mathcal{B}_{\text{strict}}(\Phi_{\text{strict}})$.*

We call the graph of Theorem 1 the *strict graph*. A compilation is *transform-wise* when it replaces each transform of the block by its own graph that computes that transform exactly; the graphs then compose. The strict graph is transform-wise: Lemmas 3–6 in Methods give the graph of each transform.

Strict compilation pays an unavoidable pointwise cost

A multiply–accumulate (MAC) array computes products of matrices and reuses each loaded number many times. A unary map takes one number and returns one, so it runs on a separate, slower scalar, vector or lookup-table unit. We count the two kinds of work separately.

Definition 4 (Pointwise, contraction and added work)**.** For a graph G, the pointwise work $\mathrm{PW}(G)$ is the number of scalar evaluations of unary maps in one forward pass; it counts evaluations, not nodes. The contraction work $\mathrm{MAC}(G)$ is the number of multiply–accumulate steps in its fixed affine nodes and in its products of two activations. The *source graph* of a transform T is the graph of its defining formula in Definition 3, with each product of two activations as one product node; $\mathrm{PW}(T)$ and $\mathrm{MAC}(T)$ are its work. Score is one product node with $\mathrm{PW} = 0$ and $\mathrm{MAC} = N^2 d$ per head, LayerNorm forms its second moment with a product node and pays one inverse square root per token, and softmax pays one exponential per entry and one reciprocal per row. If a transform-wise compilation replaces T by the graph G_T, its added work is $\Delta\mathrm{PW}(T) = \mathrm{PW}(G_T) - \mathrm{PW}(T)$ and $\Delta\mathrm{MAC}(T) = \mathrm{MAC}(G_T) - \mathrm{MAC}(T)$.

Theorem 2 (Lower bound for product-free compilation)**.** *Fix one attention head and any set Φ of scalar functions, and let the entries of Q, K, P and V range freely over $\mathbb{R}$. Every product-free graph that computes* $\mathrm{Score} = QK^\top/\sqrt{d}$ *exactly has* $\mathrm{PW} \geq N^2$*, and every product-free graph that computes* $\mathrm{Mix} = PV$ *exactly has* $\mathrm{PW} \geq Nd$*.*

The source graphs of Score and Mix make no unary evaluation, so the bounds hold for the added work $\Delta\mathrm{PW}$. They hold for every depth, width and choice of affine coefficients, and they count evaluations, so fusing nodes does not lower them. On inputs where its routing is fixed, a product-free graph is an affine function of its unary outputs (Methods, Lemma 9). It therefore needs at least as many unary evaluations as its target has coordinates that stay linearly independent modulo affine functions; for Score these are all N^2 entries.

The strict graph pays $N^2 d + 2Nd$ unary evaluations per head in Score and $N^2 d + N^2 + Nd$ in Mix, a factor $d + 2d/N$ above the bound of Theorem 2 for Score and $N + N/d + 1$ above it for Mix. Supplementary Note, "Tightness and scope of the bound" shows how far the bound of Theorem 2 can be raised.

For one DeiT-Tiny block with dense attention (Table 1), the strict graph makes 61 times the unary evaluations of the source block, and 92% of them fall in Score and Mix.

Activation–activation products remove the cost

Theorem 2 names the missing operation type: the product of two activations. Every NPU we survey provides it as an elementwise product, together with the sigmoid (Supplementary Table S1).

Definition 5 (Hadamard basis)**.** The Hadamard basis $\mathcal{B}_\odot(\Phi)$ has the three node types of $\mathcal{B}_{\mathrm{strict}}(\Phi)$ and a fourth, the *Hadamard product* $a \odot b$ of two activations with broadcasting.

Definition 6 (Hadamard unary set)**.** $\Phi_\odot = \{\sigma,\ (\cdot)^{-1/2},\ \mathrm{GELU}\}$.

With the Hadamard product, the square is $x \odot x$, the exponential is $e^t = \sigma(t) \odot \big(\sigma(-t)^{-1/2} \odot \sigma(-t)^{-1/2}\big)$, and the reciprocal is $x^{-1/2} \odot x^{-1/2}$.

Proposition 1 (Sigmoid and inverse square root suffice with the Hadamard product)**.** *Every finite Transformer block over $\mathcal{T}_{\mathrm{Tr}}$ is computed exactly by a finite graph over $\mathcal{B}_\odot(\Phi_\odot)$ in which softmax adds $\Delta\mathrm{PW} = 2N^2$ per head and every other transform adds $\Delta\mathrm{PW} = 0$.*

We call this graph the *Hadamard graph*. Its products add $\Delta\mathrm{MAC} \leq N^2 d$ per head to Score and Mix and $2N^2 + N$ to softmax. For one DeiT-Tiny block it makes 5.02×10^5 unary evaluations, against 1.64×10^7 for the strict graph (Table 1).

Table 1. Pointwise work per DeiT-Tiny block. Unary evaluations per instance of each transform; $N = 197$ tokens, $h = 3$ heads of dimension $d = 64$, MLP width $4hd$. Source: the source graph (Definition 4), with GELU and dense attention. Bound: Theorem 2 for dense scores, Corollary 2 for the k routed scores. Strict: Theorem 1; routed: Proposition 5, with $k = 16$ in the DeiT-Tiny rows. Hadamard: Proposition 1. Extended: Theorem 3. Block totals count both LayerNorms.

Transform	Source	Bound, dense / routed	Strict	Strict, routed	Hadamard	Extended
Symbolic						
LN	N		$2Nhd{+}2N$	$2Nhd{+}2N$	N	N
softmax	$hN^2{+}hN$		$8hN^2{+}2hN$	$8hNk{+}2hN$	$3hN^2{+}hN$	$hN^2{+}hN$
$\mathrm{MLP}_{\mathrm{GELU}}$	$4Nhd$		$4Nhd$	$4Nhd$	$4Nhd$	$4Nhd$
Score	0	hN^2 / hNk	$hN^2d{+}2hNd$	$hNkd{+}2hNd$	0	0
Mix	0	hNd / hNd	$hN^2d{+}hN^2{+}hNd$	$hNkd{+}hNk{+}hNd$	0	0
DeiT-Tiny						
LN	197		7.60×10^4	7.60×10^4	197	197
softmax	1.17×10^5		9.33×10^5	7.68×10^4	3.50×10^5	1.17×10^5
$\mathrm{MLP}_{\mathrm{GELU}}$	1.51×10^5		1.51×10^5	1.51×10^5	1.51×10^5	1.51×10^5
Score	0	1.16×10^5 / 9456	7.53×10^6	6.81×10^5	0	0
Mix	0	3.78×10^4 / 3.78×10^4	7.61×10^6	6.52×10^5	0	0
Block total	2.69×10^5		1.64×10^7	1.71×10^6	5.02×10^5	2.69×10^5

The $2N^2$ evaluations of softmax come from building the exponential from sigmoids. With the exponential and the reciprocal as unary maps they vanish.

Definition 7 (Extended unary set)**.** $\Phi_{\mathrm{ext}} = \{\exp,\ (\cdot)^{-1},\ (\cdot)^{-1/2},\ \mathrm{GELU}\}$.

Φ_{ext} is the set of scalar functions in the defining formulas of $\mathcal{T}_{\mathrm{Tr}}$: the exponential and the reciprocal of softmax, the inverse square root of LayerNorm and the MLP activation.

Proposition 2 (The Hadamard product suffices)**.** *Every finite Transformer block over $\mathcal{T}_{\mathrm{Tr}}$ is computed exactly by a finite graph over $\mathcal{B}_{\odot}(\Phi_{\mathrm{ext}})$ with $\Delta\mathrm{PW}(T) = 0$ for every transform T. For* Score *and* Mix *a separate reduction node adds $\Delta\mathrm{MAC} \le N^2d$ per head; for every other transform $\Delta\mathrm{MAC}(T) = 0$.*

Matrix multiplication of two activations keeps the N^2d products of $QK^\top$ and PV, which a Hadamard product writes to memory, inside the MAC array (Supplementary Note, "Why the bound counts pointwise work").

Definition 8 (Extended basis)**.** The extended basis $\mathcal{B}_{\mathrm{ext}}(\Phi)$ adds to $\mathcal{B}_{\odot}(\Phi)$ matrix multiplication (MatMul) and batched matrix multiplication (BMM) of two activations. With $\odot$ they form the *activation–activation products*. For a common Φ, $\mathcal{B}_{\mathrm{strict}}(\Phi) \subset \mathcal{B}_{\odot}(\Phi) \subset \mathcal{B}_{\mathrm{ext}}(\Phi)$.

Theorem 3 (Exact compilation with no added work)**.** *Every finite Transformer block over $\mathcal{T}_{\mathrm{Tr}}$ is computed exactly by a finite graph over $\mathcal{B}_{\mathrm{ext}}(\Phi_{\mathrm{ext}})$, and this compilation has $\Delta\mathrm{PW}(T) = \Delta\mathrm{MAC}(T) = 0$ for every transform T.*

We call the graph of Theorem 3 the *extended graph*. It is the source graph of each transform (Definition 4), so it adds no work: $QK^\top$ and PV stay one matrix-multiplication node each, with N^2d multiply–accumulate steps per head and no unary evaluation. Theorems 2 and 3 together make activation–activation products necessary and sufficient for a transform-wise compilation that adds no pointwise work.

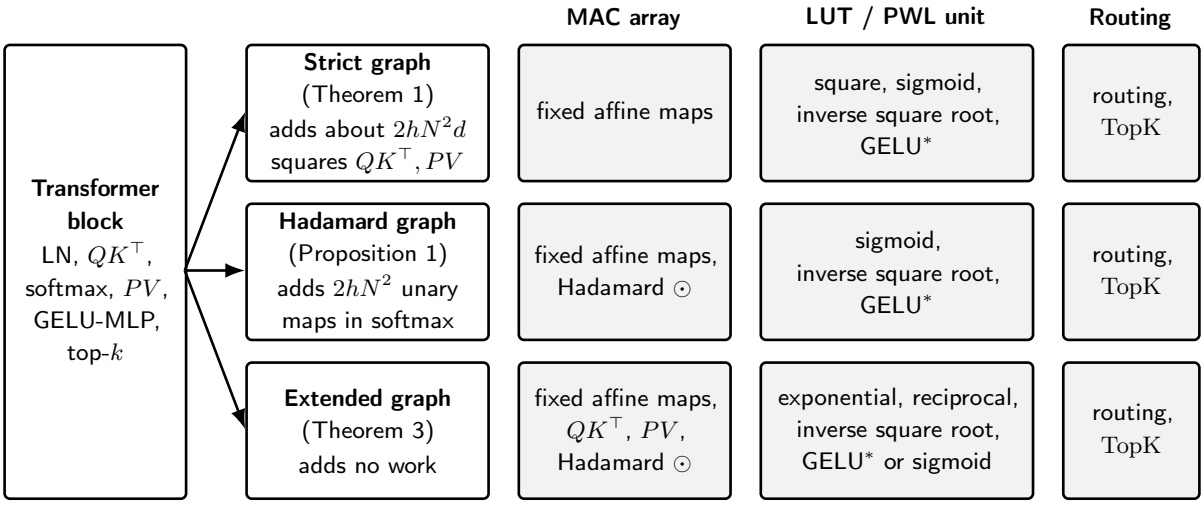


Figure 1. Three exact compilations of a Transformer block. Each row lists the nodes of one graph under the neural processing unit (NPU) component that runs them; N tokens, h heads of dimension d. MAC: multiply–accumulate; LUT: lookup table; PWL: piecewise linear; LN: layer normalization; GELU: Gaussian error linear unit; MLP: multilayer perceptron. TopK selects the k largest scores. *if used in the source graph.

To turn the bound of Theorem 2 into time, let R_{mac} and R_{pw} be the throughputs of the MAC array and the pointwise unit, and $\kappa = R_{\mathrm{mac}}/R_{\mathrm{pw}}$.

Corollary 1 (Time separation)**.** *Fix an h-head block. Assume that unary maps run only on the pointwise unit, at rate at most R_{pw}, and that the extended graph runs* Score *and* Mix *on the MAC array at rate R_{mac}. In every transform-wise product-free compilation, the* Score *and* Mix *subgraphs take at least $h(N^2 + Nd)/R_{\mathrm{pw}}$; in the extended graph they take $2hN^2d/R_{\mathrm{mac}}$. The first time is longer whenever $\kappa > 2Nd/(N + d)$, in particular whenever $\kappa \geq 2d$.*

The strict graph makes more than $2hN^2d$ unary evaluations in Score and Mix, so on them it is at least κ times slower than the extended graph. For DeiT-Tiny ($N = 197$, $d = 64$) the threshold $2Nd/(N + d)$ is 96.6. The extended basis maps onto a current NPU without a Transformer-specific unit; Figure 1 shows the unit that runs each node.

Attention weights cluster on a few keys

Fix one query q_i, the keys $k_1, \ldots, k_N$ and the values $v_1, \ldots, v_N$, the rows of K and V, and $1 \leq k < N$. Under a mask, N counts the valid keys of the row. The scores are $s_j = q_i \cdot k_j/\sqrt{d}$, sorted as $s_{(1)} \geq \cdots \geq s_{(N)}$, the attention distribution is $\pi = \mathrm{softmax}(s)$, and the attention output is $o = \sum_j \pi_j v_j$. Top-k attention keeps the support $I = \mathrm{TopK}(s, k)$ and drops the other $m = N - k$ keys, the set J. The tail mass $\tau = \sum_{j \in J} \pi_j$ is the attention outside I, and $\Delta_k = s_{(k)} - s_{(k+1)}$ is the score gap. The attention weights cluster on I when Δ_k is large. Queries cluster too.

Lemma 1 (Score gap)**.** $\tau \leq \frac{m}{k}\, e^{-\Delta_k}$.

Lemma 2 (Shared support)**.** *Let $c \in \mathbb{R}^d$ and $\rho \geq 0$. If $\|q_i - c\| \leq \rho$ and $\Delta_k > 2\rho \max_j \|k_j\|/\sqrt{d}$, the scores $c \cdot k_j/\sqrt{d}$ have the same top k keys I as q_i.*

By Lemma 2, queries within ρ of one point c share their top k keys when each meets the gap condition, and by Lemma 1 each of them puts at most $\frac{m}{k} e^{-\Delta_k}$ of its attention outside these keys.

Top-k attention with a tail-mass estimate

The softmax over the kept scores, $\tilde{\pi}_j = \exp(s_j)/\sum_{l\in I}\exp(s_l)$ for $j \in I$, needs only s_I. The dense distribution on I is $\pi_j = (1-\tau)\tilde{\pi}_j$. Write $a = \sum_{j\in I}\tilde{\pi}_j v_j$ for the mean of the kept values under $\tilde{\pi}$, $\bar{u} = \frac{1}{m}\sum_{j\in J} v_j$ for the plain mean of the dropped values, and $b = \frac{1}{\tau}\sum_{j\in J}\pi_j v_j$ for their mean under π; $\tau > 0$ since every softmax weight is positive. Then $o = (1-\tau)a + \tau b$.

Definition 9 (Tail-mass top-k attention)**.** For a tail mass $q \in [0,1]$, the estimate $\hat{p}$ of π and the output $\hat{o}$ are

$$\hat{p}_j = (1-q)\,\tilde{\pi}_j \quad (j \in I), \qquad \hat{p}_j = \frac{q}{m} \quad (j \in J), \qquad \hat{o} = (1-q)\,a + q\,\bar{u}.$$

Calibrated mode uses one constant q per block, the mean tail mass of the reference model over calibration data. *Exact* mode uses $q = \tau$. *Tail drop*, the usual top-k attention, uses $q = 0$. For $k = 0$, $I = \varnothing$, $m = N$ and $q = 1$, so $\hat{p}_j = 1/N$ and $\hat{o}$ is the mean of the values; calibrated and exact mode coincide. For $k \geq N$ the map is dense attention.

The sparse transform set $\mathcal{T}_{k,q}$ is $\mathcal{T}_{\mathrm{Tr}}$ with softmax and value mixing replaced by this map. This changes the model. Its source graph, used to count work, computes $\hat{o}$ in the all-values form

$$\hat{o} = \sum_{j\in I}\Big(\hat{p}_j - \frac{q}{m}\Big)v_j + \frac{q}{m}\sum_{j=1}^{N} v_j, \tag{1}$$

which reads only the kept values and the sum of all values. Value mixing then costs Nkd multiply–accumulate steps per head instead of $N^2 d$, plus the sum of all values, kept as a prefix sum under a causal mask. Exact mode also forms all N^2 scores for the dense normalizer.

Proposition 3 (Sparse attention compiles exactly)**.** *Every finite block over $\mathcal{T}_{k,q}$ is computed exactly by a finite graph over $\mathcal{B}_{\mathrm{strict}}(\Phi_{\mathrm{strict}})$, and by a finite graph over $\mathcal{B}_{\mathrm{ext}}(\Phi_{\mathrm{ext}})$ with $\Delta\mathrm{PW}(T) = \Delta\mathrm{MAC}(T) = 0$ for every transform T.*

Let $O = (I, s_I, V)$ be what top-k attention reads: the kept keys, their scores and the values.

Theorem 4 (Tail-mass estimate is unbiased and optimal)**.** *Let the scores s and the values V be random, with $\mathbb{E}\|v_j\|^2 < \infty$ for every j. Assume that the tail scores $(s_j)_{j\in J}$ are exchangeable given O: their conditional law is invariant under permutations of J.*

(i) With $q = \mathbb{E}[\tau \mid O]$, the estimate satisfies $\hat{p} = \mathbb{E}[\pi \mid O]$ and $\hat{o} = \mathbb{E}[o \mid O]$. Both are unbiased, and they minimize $\mathbb{E}\|\hat{p} - \pi\|^2$ and $\mathbb{E}\|\hat{o} - o\|^2$ among all functions of O. In exact mode the same holds given (O, τ), with $q = \tau$.

(ii) Tail drop has conditional bias $\mathbb{E}[\hat{p} - \pi \mid O] = \mathbb{E}[\tau \mid O]\,(\tilde{\pi}_I, -\mathbb{1}_J/m)$ and $\mathbb{E}[\hat{o} - o \mid O] = \mathbb{E}[\tau \mid O]\,(a - \bar{u})$. Its conditional distribution bias vanishes only when $\mathbb{E}[\tau \mid O] = 0$.

Theorem 4 concerns the attention distribution and the output of one head, not the loss of the whole model. Part (i) covers exact mode, and calibrated mode when τ is mean-independent of O. Among constant tail masses, $\mathbb{E}(q-\tau)^2 = (q - \mathbb{E}\tau)^2 + \mathrm{Var}\tau$ is minimal at $q = \mathbb{E}\tau$, which the calibrated q estimates.

Proposition 4 (Error decomposition)**.** *For every query,*

$$\hat{o} - o = \tau\,(\bar{u} - b) + (q - \tau)\,(\bar{u} - a), \qquad \|\hat{p} - \pi\|_2^2 = (q-\tau)^2\,w + \sum_{j\in J}\Big(\pi_j - \frac{\tau}{m}\Big)^2,$$

with $w = \|\tilde{\pi}_I\|_2^2 + \frac{1}{m}$. Tail drop has output error $\tau\,(a - b)$.

Proposition 4 needs no assumption. The error splits into a uniform-tail term and a term from the error in the tail mass, and exact mode removes the second term, so for every query it minimizes the distribution error over q. When the tail is uniform, $\pi_j = \tau/m$ on J, so $b = \bar{u}$ and exact mode has no error, while tail drop keeps the error $\tau(a - b)$. In the expected distribution error, only $\mathbb{E}[(q-\tau)^2 w]$ depends on q, so the best constant is $q^* = \mathbb{E}[w\tau]/\mathbb{E}[w]$, which equals $\mathbb{E}\tau$ when w and τ are uncorrelated.

Centroids factor the selection scores

Exact top-k forms all N^2 scores before it selects, so its strict graph makes at least N^2 unary evaluations per head (Theorem 2). When the queries cluster, the keys can be chosen before the scores, and only the k kept scores of each query are formed.

Corollary 2 (Cost of selected scores)**.** *Fix one head and $1 \le k \le N$, and let each query select k keys by a rule that is constant on some open set of inputs. Every product-free graph that computes the Nk selected scores has* $\mathrm{PW} \ge Nk$*. Given the selected key indices, a graph over $\mathcal{B}_{\mathrm{strict}}(\Phi_{\mathrm{strict}})$ computes them with* $\mathrm{PW} = Nkd + 2Nd$*.*

For each head, k-means on calibration queries gives C centroids $c_1, \dots, c_C$, the rows of $W \in \mathbb{R}^{C\times d}$. Query i goes to its nearest centroid $c_{r(i)}$, the one that maximizes the affine score $q_i \cdot c_r - \frac{1}{2}\|c_r\|^2$, and the assignment $E \in \{0,1\}^{N\times C}$ factors the queries as $Q \approx EW$. The selection scores $EWK^\top/\sqrt{d}$, a gather of the centroid scores $WK^\top/\sqrt{d}$, replace $QK^\top/\sqrt{d}$, and the routed support of query i is the top k keys of its row. It is the exact top-k support of q_i whenever the gap condition of Lemma 2 holds for $c = c_{r(i)}$. Like TopK of the scores, it is constant on an open set of inputs, so Corollary 2 bounds its scores. The source graph of the routed map is the all-values form (1) on the routed support.

Proposition 5 (Routed support)**.** *For fixed centroids and an h-head block, tail-mass attention with a constant tail mass on the routed supports is computed exactly by a graph over $\mathcal{B}_{\mathrm{strict}}(\Phi_{\mathrm{strict}})$ whose* Score *and* Mix *subgraphs make $h(2Nkd + Nk + 3Nd)$ unary evaluations, and by a graph over $\mathcal{B}_{\mathrm{ext}}(\Phi_{\mathrm{ext}})$ with $\Delta\mathrm{PW} = 0$.*

In both graphs the assignment scores and the centroid scores are fixed affine maps with $2hCNd$ multiply–accumulate steps, and the assignment and the selection are routing, so the router adds no unary evaluation. Proposition 4 holds for every support, so the error of the routed estimate splits into the same uniform-tail and tail-mass terms.

A flocking step is one attention head

In the Motsch–Tadmor flocking model [51], agent i at position x_i with velocity v_i follows

$$\dot{v}_i = \alpha\Big(\sum_j \pi_{ij} v_j - v_i\Big), \qquad \pi_{ij} = \frac{g(\|x_i - x_j\|)}{\sum_l g(\|x_i - x_l\|)},$$

where j and l run over all agents and g is a decreasing influence function.

Proposition 6 (Flocking step)**.** *For $g(r) = e^{-\theta r^2}$ with $\theta > 0$, $\pi_{ij} = \mathrm{softmax}_j(2\theta\, x_i^\top x_j - \theta\|x_j\|^2)$, so $\sum_j \pi_{ij} v_j$ is one head of distance-based attention [52] with queries $2\theta x_i$, keys x_j, a bias $-\theta\|x_j\|^2$ per key and values v_j. The top k keys of agent i are its k nearest agents.*

Tail-mass attention (Definition 9) gives the k nearest agents of agent i the weights $(1-q)\tilde{\pi}_j$ and each of the other m agents the weight q/m. Proposition 4 gives the error of this top-k flocking step exactly, and Theorem 4 applies when the far positions are exchangeable given the kept scores, velocities and x_i. By the proof of Proposition 6, Lemma 2 holds for these scores with $\sqrt{d}$ replaced by 1, so agents close to one centroid share its routed support when the gap condition holds.

Experiments

The compiled models match their references

We compile three pretrained models with the extended graph and top-k tail-mass attention, and evaluate them before training. With all keys kept, the compiled models match their references: 91.36% against 91.36% on Pet, 72.01% against 72.02% on ImageNet, 32.51 dB against 32.51 dB on Set5, and a token negative log-likelihood (NLL) of 4.5496 against 4.5496 on WikiText-2. The strict and extended graphs compute the same model exactly (Theorems 1 and 3). Only model changes, such as top-k attention, and rounding change accuracy.

The strict graph pays the predicted cost

Table 2 measures the cost of the strict graphs of Theorem 1 and Proposition 3 on a GPU. The reference calls fused attention and LayerNorm kernels. The extended graph runs at 2.5 to 5.6 times the reference latency in Tables 2 and 4. The strict graph takes 1.2 to 2.4 times the latency and, at top-64, 2.3 times the peak memory of the extended graph. The GPU has $\kappa \approx 2$ for squares (Methods), far below the threshold of Corollary 1, so this slowdown does not come from the bound of Theorem 2. It comes from the construction: d times more pointwise work than the bound, and $N^2 d$ operands per head that the implementation writes to memory. A per-module profile puts 73 to 77% of the added time in Score and Mix (Supplementary Table S2). A merged GPU kernel avoids these writes; no transform-wise product-free graph avoids the bound of Theorem 2.

With the same GELU approximation, the strict graph stays within 0.6 points of the Hadamard graph and of its polarization variant with top-k support, and within 0.4 points routed. Two runs of one strict configuration differ by up to 0.5 points with top-k support and 1.5 points routed, since the tail mass and centroids are calibrated on augmented training images. Squares cost no accuracy beyond run-to-run variation. With top-k support, polarization makes 1.9 times the unary evaluations and, for $k \leq 64$, runs 13 to 14 times slower with 5.5 times the memory (Supplementary Note, "Polarization and squares"). In integer arithmetic both identities are exact (Methods).

The elementwise product removes most of the strict cost

With top-k support and $k \leq 64$, the Hadamard graph of Proposition 1 runs at 1.2 to 1.3 times the latency of the extended graph, against 1.4 to 1.5 for the strict graph, at similar memory (Table 2). Its measured unary counts equal Table 1: 2.2×10^6 per image at $k = 16$, against 1.1×10^8 for the strict graph. Its remaining cost is the $N^2 d$ products of $QK^\top$ per head, which this implementation writes to memory (Supplementary Table S2). With top-k support, the approximation of GELU costs the Hadamard graph 0.7 to 2.6 and the strict graph 0.3 to 2.3 points; with GELU, the Hadamard graph stays within 0.1 points of the extended graph.

Table 2. Measured cost of the compilations. DeiT-Tiny on Pet, bf16, before distillation. Accuracy (%), and latency and peak GPU memory relative to the reference, which takes 2.20 ms per batch of 32 and 62 MB (Methods). Hadamard: Proposition 1. Strict: Theorem 1; polarization: polarization for every product. The extended graph uses GELU, the others $x \odot \sigma(1.702x)$. Top-k: the k largest exact scores. Routed: k keys chosen by 32 query centroids before the scores (Methods). At $k = 197$ both supports hold every key.

	Extended			Hadamard			Strict			Strict, polarization		
k	Acc.	Latency	Memory	Acc.	Latency	Memory	Acc.	Latency	Memory	Acc.	Latency	Memory
Top-k support												
16	86.24	3.28×	1.62×	83.51	4.18×	8.72×	84.08	5.04×	8.65×	83.95	68.3×	47.1×
32	89.23	3.89×	2.32×	87.46	4.73×	8.72×	87.98	5.37×	8.65×	87.71	76.1×	47.1×
64	90.35	4.88×	3.70×	89.59	5.75×	8.72×	89.78	7.14×	8.65×	89.70	93.1×	47.1×
197	91.36	5.41×	2.45×	90.35	10.5×	17.3×	90.73	13.0×	17.2×	90.57	132×	63.1×
Routed support												
16	80.13	3.35×	1.54×	78.60	3.48×	2.21×	78.22	4.73×	2.19×	78.44	14.1×	5.90×
32	87.14	4.03×	2.28×	85.66	4.10×	3.56×	85.96	4.73×	3.55×	85.55	22.3×	11.0×
64	89.81	4.37×	3.73×	88.63	5.56×	6.27×	88.88	6.67×	6.23×	88.99	40.1×	21.2×

NPUs provide the Hadamard basis

Supplementary Table S1 shows where each graph runs on ten edge accelerators by documented operator support, with the approximation of GELU where GELU is missing. The R-Car V4H runs both graphs with its DSP. Of the other nine, the strict graph runs entirely on seven, including the Edge TPU and, in INT8, the Jetson DLA; the open-source NVDLA IP and Ethos-N78 lack the inverse square root. The extended graph runs entirely on five; the toolchains of the other four have no matrix product of two activations and place it on the host. All ten multiply two activations elementwise, so none is product-free. The Hadamard graph uses the operations of the strict graph, so it runs wherever the strict graph runs.

Standard quantization applies to the compiled graph

We apply the post-training quantization (PTQ) used for convolutional networks [53, 54]: symmetric max-abs scales from calibration data, with no training or reconstruction and no Transformer-specific quantizer. The only fitted quantities are least-squares output maps (Methods).

The strict graph in Table 3 replaces GELU by $x\,\sigma(1.702x)$, so it uses the reduced strict basis $\mathcal{B}_{\text{strict}}(\{(\cdot)^2, \sigma, (\cdot)^{-1/2}\})$. The extended graph keeps GELU and the matrix products of $\mathcal{B}_{\text{ext}}(\Phi_{\text{ext}})$. Quantizers cut each graph into units with integer inputs and outputs (Methods). Weights, the logits and the inputs of linear, convolution and product nodes take 8 bits; the fixed affine nodes of the strict graph take 12-bit inputs, since these include squares and sums of squares.

PTQ changes the accuracy of the strict graph by -1.47 to -0.05 points with per-channel weight scales, and by -1.06 to -0.14 with one scale per weight tensor. On the extended graph it changes accuracy by -2.5 to -3.9 points.

Table 3. Standard post-training quantization of the compiled DeiT-Tiny on Pet. Top-1 accuracy (%) of the bf16 float model and after PTQ of the whole model with per-channel or per-tensor weight scales. On the calibration data the fp32 simulation holds every integer sum exactly (Methods). Nodes: quantized nodes, the output identity included. PV: k/N. Strict: the reduced strict graph, before distillation. Extended: the Jeffreys-distilled extended graph.

					Per-channel		Per-tensor	
k	PV	Graph	Nodes	bf16	PTQ	Δ	PTQ	Δ
16	0.08	strict	380	84.08	82.61	−1.47	83.02	−1.06
32	0.16	strict	380	87.98	87.30	−0.68	87.27	−0.71
64	0.32	strict	380	89.78	89.72	−0.05	89.64	−0.14
197	1.00	strict	368	90.73	90.27	−0.46	89.75	−0.98
16	0.08	extended	247	88.58	86.13	−2.45	84.90	−3.68
32	0.16	extended	247	90.19	86.26	−3.92	86.54	−3.65
64	0.32	extended	247	90.76	87.44	−3.33	87.16	−3.60
197	1.00	extended	235	91.36	88.74	−2.62	88.50	−2.86

The tail-mass estimate beats tail drop

Image classification. Table 4 reports Oxford-IIIT Pet and ImageNet-1k with calibrated tail mass. The Jeffreys divergence D_J between the predictive distributions of the compiled and reference models falls from 0.98 at $k = 8$ to 0.043 at $k = 64$ and 2×10^{-4} with all keys. One epoch of distillation [55] against the reference with the Jeffreys divergence [56], with q trainable, brings top-64 to $90.43 \pm 0.18\%$ over five seeds. On ImageNet, without training, top-64 is within 1.19 points of the reference.

Image restoration. MambaIRv2-Light [30] is a super-resolution model that combines state-space scans with windowed attention over $N = 256$ positions. Its attention and LayerNorm modules compile as in DeiT-Tiny, and the state-space scans stay as they are. Table 5 reports $\times 4$ and $\times 2$ super-resolution on Set5 [57], before and after three epochs of L_1 fine-tuning on DIV2K [58]. Before training, exact tail mass gives the highest PSNR at every $k < N$, and top-64 is within 0.03 dB of the reference. After fine-tuning, exact mode and tail drop end within 0.06 dB of each other for $k \geq 4$.

Language modeling. Pythia-70M [31] is a decoder-only model with a causal mask, evaluated on WikiText-2 [59] with $N = 128$ tokens. Causal rows have different numbers of valid keys, so a single calibrated q does not apply, and we use exact tail mass. Table 6 shows that exact tail mass keeps the NLL within 0.01 nats of the reference down to $k = 16$, while tail drop loses 0.04 nats there.

Figure 2 compares the three modes. On DeiT-Tiny/Pet, exact mode at k beats tail drop at $2k$ for all tested $k \leq 32$; calibrated mode does so for $k \leq 16$. At equal $k = 8$, calibrated mode gains 73.8 points over tail drop. The per-head optimality of Theorem 4 does not order model accuracy: for $k \leq 4$ calibrated mode beats exact mode. On Pythia-70M, exact mode at k beats tail drop at $2k$ for every $k \leq 8$. On MambaIRv2-Light, exact mode is the best of the three at every $k < N$. Calibrated mode falls below tail drop there for $k \geq 16$, since one constant per block fits poorly a per-query tail mass with s.d. about 0.2 at a mean of 0.57 ($k = 16$).

Table 4. DeiT-Tiny compiled with calibrated tail-mass top-k. Pet: top-1 accuracy (%) of the compiled model, Jeffreys divergence D_J to the reference, accuracy after one epoch of Jeffreys distillation (mean ± s.d. over five seeds), and latency of the compiled forward pass relative to the reference (Methods). ImageNet-1k: accuracy of the compiled model without training. PV is k/N. References: 91.36% (Pet), 72.02% (ImageNet).

		Oxford-IIIT Pet				ImageNet-1k
k	PV	Compiled	D_J	Distilled	Latency	Compiled
1	0.01	15.73	6.26	41.22 ± 0.60	2.45×	10.84
2	0.01	42.95	4.07	65.30 ± 0.47	2.85×	27.79
4	0.02	67.10	2.20	78.22 ± 0.18	2.99×	45.03
8	0.04	79.50	0.97	84.91 ± 0.15	3.06×	57.20
16	0.08	86.24	0.38	88.56 ± 0.19	3.28×	64.66
32	0.16	89.23	0.14	90.18 ± 0.08	3.89×	68.81
64	0.32	90.35	0.045	90.43 ± 0.18	4.88×	70.83
96	0.49	90.76	0.016	90.86 ± 0.22	5.32×	71.50
128	0.65	91.11	0.0055	91.11 ± 0.00	5.61×	71.82
197	1.00	91.36	0.0002	91.36 ± 0.00	5.41×	72.01

Clustered queries cut the cost of the strict graph

The measured unary counts equal Proposition 5 at every tested k. At $k = 16$ the routed strict DeiT-Tiny graph uses 2.2× instead of 8.6× the reference memory and makes 2.4×10^7 instead of 1.1×10^8 unary evaluations per image (Table 2), at 78.22% against 84.08% accuracy; its latency falls from 5.0 to 4.7 times the reference. With twice the keys it matches exact top-k: at $k = 32$ it reaches 85.96% with 37% of the unary evaluations and 41% of the memory of exact top-16.

The routed support is accurate where the queries of a cluster attend to the same keys. On DeiT-Tiny it holds 83% to 95% of the attention mass of exact top-k for $k = 8$ to 64, and 99% in the last block, where the queries are most clustered. Pythia-70M queries cluster more tightly, but queries of one cluster attend to different keys, and with tail drop its perplexity rises by 9% at $k = 32$. Routing leaves the extended graph unchanged at $N = 197$ and cuts the Hadamard graph's memory by 75% at $k = 16$.

Discussion

An NPU needs no attention, softmax or normalization unit to run Transformers. The elementwise product of two activations removes the pointwise cost that Theorem 2 prices, and a MAC array that accepts two activation operands only keeps the N^2d products of $QK^\top$ and PV out of memory.

Theorem 2 applies to transform-wise compilation, which keeps the attention contractions as subgraphs; a compilation that never forms $QK^\top$ is outside it (Supplementary Note, "Tightness and scope of the bound"). Corollary 1 needs hardware whose MAC array outruns its pointwise unit by more than $2Nd/(N + d)$.

The construction extends to other blocks. RMSNorm [42] compiles as LayerNorm without the mean. A gated MLP such as SwiGLU [60] multiplies two activations, and its SiLU is $x\,\sigma(x)$, so both take a polarization graph in the strict basis and a Hadamard node in the Hadamard and

Table 5. MambaIRv2-Light on Set5. PSNR (dB) before (pre) and after (post) three epochs of L_1 fine-tuning on DIV2K, selected on DIV2K training crops (Methods). References: 32.51 dB (×4), 38.24 dB (×2). At $k = 0$ all weight sits on the uniform tail, which tail drop cannot express; at $k = 256$ the modes coincide and the tail-drop cells repeat the calibrated run.

	×4						×2	
	Tail drop		Calibrated		Exact		Calibrated	
k	pre	post	pre	post	pre	post	pre	post
0	–	–	30.22	31.10	30.22	31.10	35.98	37.24
1	22.11	30.64	30.45	31.32	30.95	31.73	36.17	37.36
2	26.05	31.63	30.62	31.55	31.36	31.97	36.35	37.52
4	29.66	32.07	30.85	31.81	31.71	32.11	36.50	37.62
16	31.99	32.37	31.55	32.23	32.19	32.34	37.05	37.90
64	32.43	32.47	32.27	32.45	32.48	32.51	37.90	38.13
256	32.51	32.51	32.51	32.51	32.50	32.52	38.23	38.23

Table 6. Pythia-70M on WikiText-2. Token NLL (nats) and perplexity (PPL) of the compiled model without training, with exact tail mass and with tail drop, over 2.53×10^5 validation tokens in float32. $\bar{\tau}$ is the mean omitted tail mass of top-k on the dense scores of 4 random training minibatches of 4 sequences. Reference: NLL 4.5496, PPL 94.59.

	Exact tail mass			Tail drop			
k	NLL	ΔNLL	PPL	NLL	ΔNLL	PPL	$\bar{\tau}$
128	4.5496	0.0000	94.59	4.5496	0.0000	94.59	0.000
64	4.5497	0.0002	94.61	4.5497	0.0002	94.61	0.004
32	4.5513	0.0017	94.75	4.5567	0.0072	95.27	0.023
16	4.5584	0.0088	95.43	4.5885	0.0390	98.35	0.062
8	4.5822	0.0326	97.73	4.6811	0.1316	107.89	0.123
4	4.6534	0.1038	104.94	4.9232	0.3737	137.44	0.205
2	4.8267	0.2771	124.80	5.5418	0.9923	255.14	0.313
1	5.1804	0.6308	177.76	6.6381	2.0885	763.65	0.440

extended bases.

For autonomous and multi-agent systems, the Hadamard graph lets a robot or drone run a Transformer policy on an NPU built for convolutional networks. By Proposition 6, top-k attention is the topological rule of starling flocks, where each bird follows a fixed number of nearest neighbors [26], and the tail-mass term describes the far agents by their total weight and mean velocity, as cluster-level models of multi-agent control do [61].

Future work. The latency and memory reported here are GPU measurements, and the NPU placement of Supplementary Table S1 comes from documented operator support. Compiling the three graphs and their quantized integer units with the NPU toolchains, and running them on the devices, is the next step. A per-query predictor of the tail mass from the selected scores would bring calibrated mode toward exact mode without the dense normalizer. A swarm optimizer built on top-k attention with a tail-mass term is the multi-agent counterpart of these results.

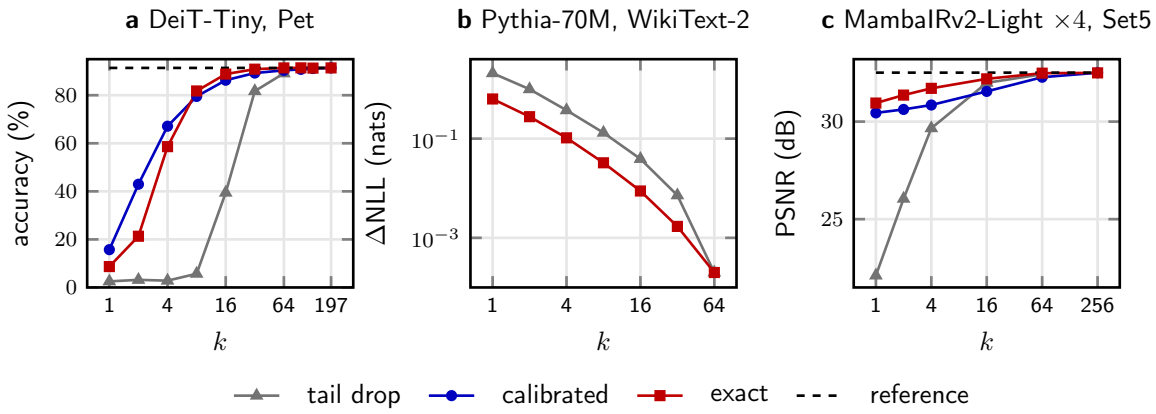


Figure 2. Tail-mass estimate against tail drop. Quality of the compiled model before any training, as a function of the number of kept keys k. (a) DeiT-Tiny on Oxford-IIIT Pet, top-1 accuracy. (b) Pythia-70M on WikiText-2, increase of the token negative log-likelihood (NLL) over the reference ($N = 128$); calibrated mode does not apply to causal rows, whose numbers of valid keys differ. (c) MambaIRv2-Light ×4 super-resolution on Set5, peak signal-to-noise ratio (PSNR). Calibrated tail mass is estimated on training data. Exact tail mass uses the dense normalizer and serves as an accuracy-evaluation reference. Tail drop discards the omitted mass.

Methods

Notation

$\langle \mathcal{B} \rangle$ is the class of finite acyclic graphs over a basis $\mathcal{B}$. It is closed under composition. We write $F \in \langle \mathcal{B} \rangle$ when a graph in $\langle \mathcal{B} \rangle$ computes F exactly. $M(x, y)$ is the polarization graph of xy: one fixed affine node forms $x \pm y$, two squares follow, and one fixed affine node returns $\frac{1}{4}[(x+y)^2 - (x-y)^2] = xy$. $\tilde{M}(x, y)$ is the norm-expansion graph of xy when x^2 and y^2 are available: one fixed affine node forms $x + y$, one square follows, and one fixed affine node returns $\frac{1}{2}[((x+y)^2 - x^2) - y^2] = xy$, subtracting the two close squares first. For an index tuple I, $\mathrm{gather}(s, I) = (s_{i_1}, \ldots, s_{i_k})$. TopK returns indices in descending order of score with a fixed tie-breaking rule.

Compilation lemmas

Each lemma realizes one transform. In this section $\mathcal{B}_{\mathrm{strict}}$ is $\mathcal{B}_{\mathrm{strict}}(\Phi_{\mathrm{strict}})$ (Definition 2) and $\mathcal{B}_{\mathrm{ext}}$ is $\mathcal{B}_{\mathrm{ext}}(\Phi_{\mathrm{ext}})$ (Definition 7).

Lemma 3 (Affine maps, residuals, routing, GELU MLP)**.** *Fixed affine maps, residual sums and routing maps are nodes of both bases. $x \mapsto A_2\mathrm{GELU}(A_1 x + b_1) + b_2$ is the composition $\mathrm{Aff}_{A_2,b_2} \circ \mathrm{GELU} \circ \mathrm{Aff}_{A_1,b_1}$ in both bases.*

Proof. For the approximation, $x\,\sigma(1.702x)$ is one sigmoid after a fixed scaling and one product: $M\big(x, \sigma(1.702x)\big)$, two squares per hidden unit, in the strict basis, and one Hadamard node in the extended basis. □

Lemma 4 (LayerNorm)**.** $\mathrm{LN} \in \langle \mathcal{B}_{\mathrm{strict}} \rangle$ *and* $\mathrm{LN} \in \langle \mathcal{B}_{\mathrm{ext}} \rangle$.

Proof. Let $u = x - \mathrm{mean}(x)\mathbb{1}$, which is affine. Strict: $r^2 = \mathrm{mean}_i(u_i^2)$ uses squares and an affine mean, $a = (r^2 + \varepsilon)^{-1/2}$ is one unary evaluation per token, and $z_i = \tilde{M}(u_i, a) = u_i/\sqrt{r^2 + \varepsilon}$ reuses u_i^2 and forms a^2 once per token, so it costs one square per entry and one per token. Then

$\mathrm{LN}_i(x) = \beta_i + \gamma_i z_i$ is affine in z. Extended: $r^2 = \mathrm{mean}(u \odot u)$ is a Hadamard product followed by an affine mean, $a = (r^2 + \varepsilon)^{-1/2}$ is one unary evaluation per token, and $\mathrm{LN}(x) = \beta + \gamma \odot (u \odot a)$, where the product with the fixed γ is affine and $u \odot a$ is a broadcast Hadamard product. □

Lemma 5 (Scores and value mixing)**.** *$(Q, K) \mapsto QK^\top/\sqrt{d}$ and $(P, V) \mapsto PV$ lie in $\langle \mathcal{B}_{\mathrm{strict}} \rangle$ and in $\langle \mathcal{B}_{\mathrm{ext}} \rangle$.*

Proof. Strict: $S_{ij} = (2\sqrt{d})^{-1}\big[\sum_f (q_{if} + k_{jf})^2 - \|q_i\|^2 - \|k_j\|^2\big]$ and $(PV)_{ij} = \frac{1}{2}\big[\sum_\ell \big(v_{\ell j}^2 - (v_{\ell j} - p_{i\ell})^2\big) + \sum_\ell p_{i\ell}^2\big]$, with the norms formed once per row and the sums affine: $N^2 d + 2Nd$ squares per head for the scores and $N^2 d + N^2 + Nd$ for the mix. In the mix the two close squares are subtracted per term, inside the node that sums over ℓ. Extended: each map is one MatMul or BMM node followed by a fixed scaling. □

Lemma 6 (Softmax)**.** *$s \mapsto \mathrm{softmax}(s)$ lies in $\langle \mathcal{B}_{\mathrm{strict}} \rangle$ and in $\langle \mathcal{B}_{\mathrm{ext}} \rangle$.*

Proof. Let $t_i = s_i - \max_l s_l \le 0$, where the maximum is a routing selection, and $n_i = e^{t_i}$, so that $Z = \sum_j n_j \ge 1$. The shift leaves softmax unchanged and keeps $\sigma(-t_i)$ in $[\frac{1}{2}, 1)$ and Z in $[1, N]$, so every inverse square root below acts on a number in $[\frac{1}{2}, N]$. Strict: since $\sigma(t)/\sigma(-t) = e^t$ for every t, $n_i = M\big(\sigma(t_i), (\sigma(-t_i)^{-1/2})^2\big)$, with two sigmoids, one inverse square root and three squares per entry; Z is affine in n, and $p_i = M\big(n_i, (Z^{-1/2})^2\big)$ adds two squares per entry and one inverse square root and one square per row. Extended: $n_i = \exp(t_i)$, one exponential per entry, and $p = n \odot Z^{-1}$, one reciprocal per row and a broadcast Hadamard product. □

Lemma 7 (Tail-mass top-k attention)**.** *The map of Definition 9 from (s, V) to $\hat{o}$, for a fixed calibrated q or for $q = \tau$, lies in $\langle \mathcal{B}_{\mathrm{strict}} \rangle$ and in $\langle \mathcal{B}_{\mathrm{ext}} \rangle$.*

Proof. Masks are fixed routing and affine maps. We treat three cases.

$k = 0$. Definition 9 gives $\hat{o} = \frac{1}{N}\sum_j v_j$, a fixed affine map.

$k \ge N$. Nothing is dropped and the map is dense attention, covered by Lemmas 5 and 6.

$1 \le k < N$. $I = \mathrm{TopK}(s, k)$ and $\hat{s} = \mathrm{gather}(s, I)$ are routing. In both modes the graph computes the all-values form (1), which equals Definition 9 because $\sum_{j \in J} v_j = \sum_{j=1}^N v_j - \sum_{j \in I} v_j$. Its parts are the coefficients $\omega_j = \hat{p}_j - q/m$, the kept mixing $\sum_{j \in I} \omega_j v_j$, and the tail term $(q/m)\sum_{j=1}^N v_j$. The kept mixing is the norm expansion of Lemma 5 over the k kept values in the strict basis and one product node over k values in the extended basis. In calibrated mode, $\tilde{\pi}$ follows Lemma 6 on $\hat{s}$, with the selected maximum $\hat{s}_1$ as shift; $\hat{p}_j = (1 - q)\tilde{\pi}_j$ and ω_j are affine in $\tilde{\pi}$, and the tail term is a fixed affine map of V. In exact mode, the dense normalizer is formed from all scores with the dense maximum as shift: $n_j = \exp(s_j - \max_l s_l)$, $Z = \sum_j n_j$, $\hat{p}_j = n_j/Z$ for $j \in I$ (Lemma 6), and $\tau = 1 - \sum_{j \in I} \hat{p}_j$. The coefficients $\omega_j = \hat{p}_j - \tau/m$ are affine in $(\hat{p}, \tau)$. The tail term $(\tau/m)\sum_j v_j$ is a product of the scalar activation τ with the d-dimensional activation $\sum_j v_j$: a norm expansion with τ^2 once per row and the squares of $\sum_j v_j$ once per head in the strict basis, and one broadcast Hadamard node with d scalar products in the extended basis. □

Proofs of Theorem 1, Propositions 1 and 2 and Theorem 3

Proof of Theorem 1. Replace every transform of the block by its strict realization from Lemmas 3–6. Each realization computes its transform exactly, and $\langle \mathcal{B}_{\mathrm{strict}} \rangle$ is closed under composition. □

Proof of Proposition 1. Take the strict graph of Theorem 1. Replace each product graph, $M(x,y)$ or $\tilde{M}(x,y)$, by one Hadamard node $x \odot y$, drop the squares of the factors it used, and form each remaining square as $x \odot x$. Since $M(x,y) = \tilde{M}(x,y) = xy$, the new graph computes the same function, and its unary maps are those of $\Phi_{\odot}$. Score and Mix become a broadcast Hadamard product and a fixed sum, $S_{ij} = d^{-1/2} \sum_f (q_{i\cdot} \odot k_{j\cdot})_f$ and $(PV)_{ij} = \sum_\ell (p_{i\cdot} \odot v_{\cdot j})_\ell$, with no unary map. LayerNorm keeps one inverse square root per token, as its source graph. The MLP keeps one GELU per hidden unit, or one sigmoid and one Hadamard product for the approximation, as its source graph. Softmax keeps, per entry, two sigmoids and one inverse square root for $n_i = \sigma(t_i) \odot \big(\sigma(-t_i)^{-1/2} \odot \sigma(-t_i)^{-1/2}\big)$, and, per row, one inverse square root for $1/Z = Z^{-1/2} \odot Z^{-1/2}$ (Lemma 6). Its source graph pays one exponential per entry and one reciprocal per row, so softmax adds $2N^2$ unary evaluations per head. For the contraction work, the products of Score and Mix cost the $N^2 d$ multiplications of the source, and each fixed sum adds at most $N^2 d$ multiply–accumulate steps per head. Softmax adds two products per entry for n_i and one per row for $1/Z$, so $2N^2 + N$ per head; its sum over the row and its product $n \odot Z^{-1}$ are those of the source graph. □

Proof of Proposition 2. Replace every transform except Score and Mix by its extended realization from Lemmas 3–6; these use only Hadamard products among the activation–activation products. Each performs the operations of its source graph (Definition 4). Affine maps, residuals, routing and the GELU MLP are basis nodes. LayerNorm performs, per token, an affine mean, the product $u \odot u$ and its mean, one inverse square root, and the rescaling. Softmax performs one exponential per entry, an affine sum, one reciprocal per row and a broadcast product. So $\Delta\mathrm{PW} = \Delta\mathrm{MAC} = 0$ for these transforms. Compute Score and Mix as in the proof of Proposition 1, by a broadcast Hadamard product and a fixed reduction. No unary map is added, so $\Delta\mathrm{PW} = 0$, and the reduction adds at most $N^2 d$ multiply–accumulate steps per head. □

Proof of Theorem 3. Replace every transform of the block by its extended realization from Lemmas 3–6. Each realization computes its transform exactly, and $\langle \mathcal{B}_{\mathrm{ext}} \rangle$ is closed under composition. Every transform except Score and Mix has the realization of the proof of Proposition 2, with $\Delta\mathrm{PW} = \Delta\mathrm{MAC} = 0$. Score and Mix are single MatMul or BMM nodes of the source shape: $\mathrm{PW} = 0$ and $\mathrm{MAC} = N^2 d$ before and after. Hence $\Delta\mathrm{PW} = \Delta\mathrm{MAC} = 0$ for every transform. □

The bases $\mathcal{B}_{\mathrm{strict}}(\Phi_{\mathrm{strict}})$ and $\mathcal{B}_{\mathrm{ext}}(\Phi_{\mathrm{ext}})$ are not nested. With $\Phi_{\mathrm{all}} = \Phi_{\mathrm{strict}} \cup \Phi_{\mathrm{ext}}$, Theorems 1 and 3 also hold in the nested bases $\mathcal{B}_{\mathrm{strict}}(\Phi_{\mathrm{all}}) \subset \mathcal{B}_{\mathrm{ext}}(\Phi_{\mathrm{all}})$, since enlarging the unary set keeps every graph valid.

Proof of Theorem 2

The proof follows the nonlinear rank of the target. Φ is an arbitrary set of real functions. For a set $D \subseteq \mathbb{R}^n$ write $\mathcal{A}(D)$ for the restrictions to D of affine functions on $\mathbb{R}^n$. For $F\colon D \to \mathbb{R}^r$, the nonlinear rank

$$\mathrm{nrk}_D(F) = \dim\Big(\big(\mathrm{span}\{F_1, \dots, F_r\} + \mathcal{A}(D)\big)/\mathcal{A}(D)\Big)$$

counts the coordinates of F that stay linearly independent on D after adding affine functions.

Lemma 8 (Fixed routing on a set dense in a ball)**.** *Let G be a finite product-free graph with inputs in $\mathbb{R}^n$, and let $U \subseteq \mathbb{R}^n$ be a nonempty open set. There is a set $D \subseteq U$ and an open ball $B \subseteq U$ such that D is dense in B and every routing decision of G is the same at every point of D.*

Proof. G has finitely many routing nodes, and each, with its fixed tie-breaking rule, has finitely many outcomes: an order of finitely many compared values, or a selection among finitely many entries. The joint outcome therefore takes finitely many values, and the sets of points of U with a given joint outcome form a finite partition of U. A finite union of nowhere dense sets is nowhere dense, and U is not, so some part D' is not nowhere dense: its closure contains an open ball B, which we may take inside U. Then $D = D' \cap B$ is dense in B. □

No continuity is used: the outcome sets may be arbitrary, so Φ needs no continuity or measurability.

Lemma 9 (Affine reduction)**.** *Let G be product-free with outputs $G_1, \dots, G_r$, let all routing of G be constant on a set D, and let $y_1, \dots, y_u$ be the outputs of its unary evaluations, $u = \mathrm{PW}(G)$, viewed as functions on D. Then there are affine ℓ_j and constants c_{jt} with $G_j = \ell_j + \sum_t c_{jt} y_t$ on D.*

Proof. By induction along a topological order, every value in G is on D an affine function of $(x, y_1, \dots, y_u)$. Inputs are. A fixed affine node maps affine functions to affine functions. A routing node acts on D as a fixed 0/1 selection matrix. A unary node outputs some y_t. No node multiplies two values, so the induction never leaves the affine functions of (x, y). □

Lemma 10 (Pointwise work bounds nonlinear rank)**.** *If a product-free graph G computes F exactly on a set D on which its routing is constant, then* $\mathrm{PW}(G) \geq \mathrm{nrk}_D(F)$.

Proof. By Lemma 9, every F_j equals $\sum_t c_{jt} y_t$ modulo $\mathcal{A}(D)$. In the ambient quotient of all real functions on D by $\mathcal{A}(D)$, the classes of $F_1, \dots, F_r$ lie in the span of the u unary-output classes. Their span therefore has dimension at most u. □

Lemma 11 (Rank of quadratic forms)**.** *Let D be dense in an open ball $B \subseteq \mathbb{R}^n$, and let $F_1, \dots, F_r$ be homogeneous quadratic polynomials on $\mathbb{R}^n$ such that each F_s has a monomial, with nonzero coefficient, that no other F_t has. Then* $\mathrm{nrk}_D(F) = r$.

Proof. Suppose $\sum_s \lambda_s F_s$ agrees on D with an affine function. Both sides are continuous and D is dense in B, so they agree on B. Polynomials that agree on an open set agree everywhere, and a homogeneous quadratic polynomial that is affine is zero. The coefficient of the monomial that only F_s has is λ_s times a nonzero number, so every λ_s vanishes. □

Proof of Theorem 2. Let G be a product-free graph with inputs in $\mathbb{R}^n$ that computes $S = QK^\top/\sqrt{d}$ exactly, and take D and B from Lemma 8 with $U = \mathbb{R}^n$. The entry $S_{ij} = d^{-1/2} \sum_f q_{if} k_{jf}$ is a homogeneous quadratic polynomial, and no other entry has its monomials $q_{if} k_{jf}$. Lemma 11 gives $\mathrm{nrk}_D(S) = N^2$, and Lemma 10 gives $\mathrm{PW}(G) \geq N^2$. For Mix, the entry $(PV)_{ij} = \sum_\ell p_{i\ell} v_{\ell j}$, $(i, j) \in [N] \times [d]$, has the monomials $p_{i\ell} v_{\ell j}$, which no other entry has, so Lemmas 11 and 10 give $\mathrm{PW}(G) \geq Nd$. □

Proof of Corollary 1. A transform-wise product-free compilation of an h-head block contains its own Score and Mix subgraph for each head, each exact for all arguments and sharing no unary evaluation with the others, so these subgraphs have $\mathrm{PW} \geq h(N^2 + Nd)$ by Theorem 2. Unary maps run only on the pointwise unit, at rate at most R_{pw}, so they take at least $h(N^2 + Nd)/R_{\mathrm{pw}}$. In the extended graph, Score and Mix have $\mathrm{PW} = 0$ and $\mathrm{MAC} = 2hN^2 d$ and, by assumption, take $2hN^2 d/R_{\mathrm{mac}}$. The first time exceeds the second exactly when

$\kappa > 2N^2d/(N^2 + Nd) = 2Nd/(N + d)$, which is below $2d$. The strict graph of Theorem 1 has $\mathrm{PW} = h(2N^2d + N^2 + 3Nd) > 2hN^2d$ in Score and Mix (Lemma 5), hence a pointwise time of at least κ times the extended time. □

Proofs for clustered attention

Proof of Lemma 1. The normalizer is at least $\sum_{j\in I} e^{s_j} \geq k\, e^{s_{(k)}}$, and the tail sum $\sum_{j\in J} e^{s_j}$ is at most $m\, e^{s_{(k+1)}}$, so $\tau \leq \frac{m}{k} e^{s_{(k+1)} - s_{(k)}}$. □

Proof of Lemma 2. Let $R = \max_l \|k_l\|$. For every key, $|q_i \cdot k_j - c \cdot k_j| \leq \|q_i - c\|\, \|k_j\| \leq \rho R$. For $j \in I$ and $l \in J$, $(c \cdot k_j - c \cdot k_l)/\sqrt{d} \geq (q_i \cdot k_j - q_i \cdot k_l)/\sqrt{d} - 2\rho R/\sqrt{d} \geq \Delta_k - 2\rho R/\sqrt{d} > 0$, so every key of I scores above every key of J under c. □

Proof of Proposition 3. Exactness: replace softmax and value mixing by the realization of Lemma 7 in each basis, and the other transforms as in Theorems 1 and 3. Work in the extended basis: for $k = 0$ the map is an affine average and for $k \geq N$ it is dense attention, both equal to their source graphs. For $1 \leq k < N$ the source graph is the all-values form (1) with the q of the mode, and the extended realization of Lemma 7 is this graph. TopK and gather are routing. The softmax follows Lemma 6, with one exponential per kept score in calibrated mode and one per score in exact mode, and one reciprocal per row. The kept mixing is one product node over k values. The tail term is a fixed affine map in calibrated mode and one broadcast product node with d scalar products in exact mode. Hence $\Delta\mathrm{PW} = \Delta\mathrm{MAC} = 0$ for every transform in both modes. □

Proof of Theorem 4. (i) Given O, the normalizer $Z = \sum_j e^{s_j}$ is a symmetric function of the tail scores, so the tail weights $\pi_j = e^{s_j}/Z$, $j \in J$, are exchangeable given O, and $\mathbb{E}[\pi_j \mid O]$ is the same for all $j \in J$. These m values sum to $\mathbb{E}[\tau \mid O]$, so each equals $\mathbb{E}[\tau \mid O]/m$. For $j \in I$, $\pi_j = (1 - \tau)\tilde{\pi}_j$ and $\tilde{\pi}_j$ is a function of O, so $\mathbb{E}[\pi_j \mid O] = (1 - \mathbb{E}[\tau \mid O])\tilde{\pi}_j$. With $q = \mathbb{E}[\tau \mid O]$ this is $\hat{p} = \mathbb{E}[\pi \mid O]$. Since V is part of O and o is linear in π, $\mathbb{E}[o \mid O] = \sum_j \mathbb{E}[\pi_j \mid O] v_j = \hat{o}$. The weights lie in $[0, 1]$ and $\mathbb{E}\|v_j\|^2 < \infty$, so π and o are square integrable, and a conditional expectation of a square-integrable variable is unbiased and minimizes the mean squared error over all functions of the conditioning variables. In exact mode, τ is observed. Given O, τ is a symmetric function of the tail scores, so they stay exchangeable given (O, τ). Then $\mathbb{E}[\pi_j \mid O, \tau]$ is the same for all $j \in J$ and sums to τ over J, so it equals τ/m, and $\mathbb{E}[\pi_j \mid O, \tau] = (1 - \tau)\tilde{\pi}_j$ for $j \in I$. This is $\hat{p}$ with $q = \tau$. (ii) For $q = 0$, $\hat{p}_I = \tilde{\pi}_I$ and $\hat{p}_J = 0$, so $\mathbb{E}[\hat{p} - \pi \mid O] = (\tilde{\pi}_I - (1 - \mathbb{E}[\tau \mid O])\tilde{\pi}_I,\ -\mathbb{E}[\tau \mid O]\mathbb{1}_J/m) = \mathbb{E}[\tau \mid O](\tilde{\pi}_I, -\mathbb{1}_J/m)$. By (i), $\mathbb{E}[o \mid O] = (1 - \mathbb{E}[\tau \mid O])a + \mathbb{E}[\tau \mid O]\bar{u}$, and $\hat{o} = a$ for $q = 0$, so $\mathbb{E}[\hat{o} - o \mid O] = \mathbb{E}[\tau \mid O](a - \bar{u})$. □

Proof of Proposition 4. By Definition 9, $\hat{o} = (1 - q)a + q\bar{u}$, and $o = (1 - \tau)a + \tau b$. Subtracting, $\hat{o} - o = (\tau - q)a + q\bar{u} - \tau b = \tau(\bar{u} - b) + (q - \tau)(\bar{u} - a)$. For $q = 0$ this is $\tau(a - b)$. For the distribution, $\hat{p}_j - \pi_j = (\tau - q)\tilde{\pi}_j$ on I, and $\hat{p}_j - \pi_j = (q - \tau)/m + (\tau/m - \pi_j)$ on J. Squaring and summing, the cross term is $2(q - \tau)m^{-1} \sum_{j\in J}(\tau/m - \pi_j) = 0$, since $\sum_{j\in J} \pi_j = \tau$, which leaves $(q - \tau)^2(\|\tilde{\pi}_I\|_2^2 + 1/m) + \sum_{j\in J}(\pi_j - \tau/m)^2$. □

Proof of Corollary 2. Choose an open set U of inputs on which the selection is constant, take D and B from Lemma 8 inside U, and write the selected key indices of query i as $j_{i1}, \ldots, j_{ik}$. The selected score $F_{it} = d^{-1/2} \sum_f q_{if} k_{j_{it} f}$ has the monomials $q_{if} k_{j_{it} f}$, which no other selected score has, since the k keys of a query are distinct. Lemmas 11 and 10 give $\mathrm{PW} \geq Nk$. For the upper

bound, routing gathers the k keys of each query, each selected score costs d squares of $q_i + k_j$, and the norms of the N queries and the N keys cost $2Nd$ (Lemma 5). □

Proof of Proposition 5. The assignment scores $q_i \cdot c_r - \frac{1}{2}\|c_r\|^2$ and the centroid scores $c_r \cdot k_j$ are fixed affine maps of Q and K, with $hCNd$ multiply–accumulate steps each. TopK over centroids, TopK over keys for each centroid, and the gather of each query's list are routing. The selected scores cost $Nkd + 2Nd$ squares per head (Corollary 2), and the kept mixing of Lemma 7 costs $Nkd + Nk + Nd$ (Lemma 5). With a constant tail mass the tail term is a fixed affine map of V. In the extended basis the scores and the mixing are product nodes over k keys, and every unary map is one of the source graph. □

Proof of Proposition 6. $g(\|x_i - x_j\|) = e^{-\theta\|x_i\|^2}\, e^{2\theta x_i^\top x_j - \theta\|x_j\|^2}$. The first factor does not depend on j and cancels against the same factor in the sum over l, which leaves the softmax. The score of agent j equals $\theta\|x_i\|^2 - \theta\|x_i - x_j\|^2$, whose first term does not depend on j; since $\theta > 0$, the k largest scores belong to the k nearest agents. Lemma 2 holds for these scores with $\sqrt{d}$ replaced by 1: the bias of agent j enters the scores of every query equally and cancels in each comparison of its proof. □

Models and data

DeiT-Tiny. We use the timm [62] DeiT-Tiny [27] (patch 16, 224×224 input, $N = 197$). For Oxford-IIIT Pet [28] (37 classes, 3,680 training and 3,669 test images) we fine-tune the classifier head for 20 epochs on the training split and keep the epoch with the highest training accuracy, measured without augmentation; the reference reaches 91.36%. Pet has no validation split. We report its test split, and no step selects on it. For ImageNet-1k [29] we use the pretrained checkpoint (72.02%).

MambaIRv2-Light. We use the released $\times 2$ and $\times 4$ lightweight super-resolution weights [30]. The attention acts on 16×16 windows, so $N = 256$. Training uses DIV2K [58] patches of 64×64 low-resolution pixels; evaluation reports PSNR on the Y channel of Set5 [57] with the standard border crop.

Pythia-70M. We use EleutherAI's Pythia-70M [31] with eager attention on non-overlapping 128-token windows of WikiText-2 (raw) [59], 2.53×10^5 validation tokens. The partial rotary embedding multiplies each query and key by a fixed rotation for each position, an affine node.

Tail-mass calibration

Calibrated mode runs the reference model on calibration minibatches from the training split and records, for each block, the dense-softmax mass outside the top-k scores of every query. The block's q is the mean over queries. We use the full Pet training split (115 minibatches of 32), 10 ImageNet training minibatches and 8 DIV2K minibatches. Exact mode computes τ for each query at run time from the full softmax normalizer. Supplementary Note, "Tail-mass ablation on DeiT-Tiny" compares the tail modes and the calibrated with the exact tail mass.

Query-centroid routing

For each block and head, k-means (k-means++ initialization, 30 Lloyd iterations) fits 32 centroids c_r to at most 2^{16} queries from 20 random training minibatches (64 for Pythia-70M), taken after the rotary embedding. Each query goes to its nearest centroid and takes the top k keys of $c_{r(i)} \cdot k_j$, under the causal mask for Pythia-70M. Without a causal mask, each centroid

keeps its top k keys and each query takes the list of its centroid, so no $N \times N$ matrix is formed. For causal routing, the implementation materializes an $N \times N$ matrix of centroid selection scores and forms valid-value means with a dense lower-triangular contraction; keeping each centroid's top-k keys and the value sum over successive prefixes removes both dense operations. In calibrated mode and with tail drop the graph gathers the k routed keys of each query and forms only their scores, by the norm expansion in the strict graph and by a batched product in the extended graph. Exact mode forms all scores for its normalizer. The tail term is that of Definition 9. For the strict graph with GELU at $k = 16$, 32 and 64, the latency benchmark gives 4.19, 3.89 and 5.41 times the reference latency with the routed support against 4.34, 4.58 and 6.19 with exact top-k; the extended graph gives 3.35 to 4.37 against 3.28 to 4.88, and its peak memory changes by less than 5%. Calibrated mode uses the mean mass outside the routed support on the calibration minibatches of the top-k runs. We report, on the held-out Pet test split and the WikiText-2 validation split, the dense mass inside the routed support relative to the exact top-k support, and the accuracy before fine-tuning. Pythia-70M has no calibrated mode, so its perplexity is compared with tail dropping in both supports. With 8 and 128 centroids the routed mass at $k = 16$ is 76% and 92% of the top-k mass on DeiT-Tiny and 89% and 93% on Pythia-70M.

Distillation and fine-tuning

For DeiT-Tiny on Pet we distill [55] the compiled model against the frozen reference for one epoch with AdamW (batch 64, learning rate 5×10^{-5}, weight decay 0.05, cosine schedule without warm-up, gradient clipping at 1.0) and loss $(1 - \lambda)\,\mathrm{CE}(s, y) + \lambda\big(\mathrm{KL}(s\|t) + \mathrm{KL}(t\|s)\big)$, with $\lambda = 1$ (Jeffreys) or $\lambda = 0$ (cross-entropy), over seeds 42 to 46. Supplementary Note, "Distillation loss" compares the two losses. Each run keeps the epoch, or the model before training, with the highest training accuracy, measured without augmentation, and later steps use the seed with the highest such accuracy; ties keep the earlier one. The tail mass q is trainable, a scalar parameter per block clamped to $[0, 1]$ in the forward pass. For MambaIRv2-Light we fine-tune for three epochs with AdamW (batch 8, learning rate 2×10^{-4}, weight decay 10^{-4}, cosine schedule, gradient clipping at 1.0) on the L_1 loss to the high-resolution targets and keep the epoch with the highest training PSNR on one center crop of each DIV2K training image: the third epoch in 22 of the 26 runs, the second in 3, and the untrained model for $\times 2$ at $k = 256$. Set5 is only evaluated. Pythia-70M is evaluated without training.

Post-training quantization

Precision. We target edge NPUs, so DeiT-Tiny and MambaIRv2-Light run in bf16, which stores each value in half the memory of fp32 with 8 significant bits. With all keys, bf16 changes the accuracy of every DeiT-Tiny graph by at most 0.46 points and the PSNR of MambaIRv2-Light by at most 0.04 dB against fp32 (Supplementary Note, "bf16 against fp32"). An 8-bit PTQ code drops the 8-bit exponent of a bf16 value and keeps its sign and 7 more bits: all values of a tensor share one scale. Near the largest value of its tensor the code is as fine as bf16, and smaller values lose bits. A square doubles the exponent range of its input: a value at 2^{-4} of the largest has a square at 2^{-8} of the largest square. Bit widths are those of the input codes. The fixed affine nodes of the strict graph read squares and sums of squares, so their inputs take 12 bits; weights, the logits and the inputs of linear, convolution and product nodes, including the two scalings by the calibrated tail mass in each block, take 8 bits.

Quantized graph. Quantized nodes are the linear and convolution layers, the products of two

activations or of an activation and a parameter, and the fixed affine nodes with a coefficient tensor, including the residual and position-embedding additions. Each quantized node has one quantizer per input, and this quantizer is also the output quantizer of the preceding node, so every boundary is quantized once. An identity node at the model output gives the last layer an output quantizer too, so the logits are 8-bit codes with one scale, and the model is quantized from the image to the logits. Unary maps, routing, and plain sums and means over an axis have no quantizer of their own. A sum or mean reads integer codes with fixed scales, or their squares, so its result is an integer with a fixed scale. It keeps its full integer range, with no rounding to a specific bit width; the next quantizer sets the width. The graph between two quantizers is thus a unit with integer inputs and integer outputs: a quantized node with its output map, followed by any unary, routing and summing nodes. Every tensor that passes from one unit to the next is an integer code, so the quantized graph is an integer graph however its units are evaluated. On an NPU, a unit is an integer multiply–accumulate followed by a requantization, a fixed multiplier and offset [53], and the lookup-table or piecewise-linear units of the NPU approximate its unary nodes over the calibrated input range [48, 49].

Simulation and calibration. We simulate each unit in fp32 [54], with the integer codes held in fp32 tensors. fp32 holds every integer up to $2^{24} \approx 1.7 \times 10^7$, so the integer sums are exact. The run checks this for every node with integer inputs: over the calibration data, the sum of the absolute products of the node stays below 4.5×10^6, a factor 3.7 below this limit. The output maps and unary nodes run in fp32. The bf16 float model is the reference. Calibration runs it for two forward passes over the full Pet training split: the first records the maximum absolute value of every node input, the second fits the output maps to its outputs. Activations use one symmetric max-abs scale per input tensor. Linear and convolution weights use one symmetric scale per output channel; affine coefficient tensors use one scale. A linear or convolution node takes its output scale as the product of input and weight scales and fits a per-channel output bias; other nodes fit an affine output map by least squares. A node that combines several inputs of different range, as the norm expansion does with v^2 and $(v-p)^2$ or with u^2 and a^2, quantizes each input on its own grid and rescales it before the sum, as an integer add with per-input multipliers does. Quantizing the logits changes accuracy by -0.16 to $+0.30$ points. Supplementary Note, "Quantization with exact GELU" gives PTQ of the strict graph with exact GELU.

Integer products from squares. For integer operands both identities hold exactly when no intermediate step overflows or requantizes. For signed 8-bit operands in $[-128, 127]$, the sums and differences lie in $[-256, 255]$. A shared table of squares needs 512 signed-index entries, or 257 magnitude-index entries, with 17-bit unsigned outputs, and the difference of two squares fits in 18 signed bits. Dividing $(a+b)^2 - (a-b)^2$ by four, or $(a+b)^2 - a^2 - b^2$ by two, before accumulation gives the integer product exactly. Each product has magnitude at most 2^{14}, so a signed 32-bit accumulator holds a dot product of every length $d < 2^{17}$.

Timing and unary counts

Latency is the GPU time of the forward pass alone, divided by that of the reference, on one NVIDIA RTX 4070 Ti SUPER. Ten batches of 32 images from the held-out Pet test split are first stored on the GPU. Each model runs in bf16 under `torch.compile` in the default, reduce-overhead and max-autotune modes, and we report its fastest mode: the median of five timed runs over the ten batches, with CUDA events after a warm-up. Peak memory is the maximum allocated GPU memory during an eager evaluation pass over the held-out Pet test split, with only the measured model on the GPU; Supplementary Note, "Tensor sizes" gives the

tensor sizes behind it. Unary evaluations are counted with forward hooks on every unary node over one test minibatch. With all keys, the DeiT-Tiny count per image equals twelve block totals of Table 1 plus the final LayerNorm: 6.02×10^6 for the Hadamard graph and 1.96×10^8 for the strict graph with GELU. The implementation of the extended graph forms the squares $u \odot u$ of its 25 LayerNorms with a unary square node, so its count of 4.17×10^6 adds $25Nhd = 9.46 \times 10^5$ to that sum. With $x \odot \sigma(1.702x)$ the strict graph forms this product from two squares, $8Nhd$ more per block.

The GPU ratio. The RTX 4070 Ti SUPER has 66 Ada streaming multiprocessors at a boost clock of 2.61 GHz. Each multiprocessor retires 128 FP32 operations per clock on its CUDA cores and 256 dense bf16 multiply–accumulates per clock with FP32 accumulation on its tensor cores [63]. This gives $R_{\mathrm{pw}} \approx 22\,\mathrm{T/s}$, $R_{\mathrm{mac}} \approx 44\,\mathrm{T/s}$ and $\kappa \approx 2$.

Numerical conventions

The compiled LayerNorm uses the ε of each reference: 10^{-6} for DeiT-Tiny and 10^{-5} for MambaIRv2-Light and Pythia-70M. Softmax normalization divides by the max-shifted partition sum Z. The DeiT-Tiny and MambaIRv2-Light checkpoints store their weights in bf16; Pythia-70M runs in fp32 (Supplementary Note, "bf16 against fp32"). The bf16 runs evaluate the reference and every compiled graph under PyTorch CUDA autocast, which keeps means and sums in fp32; a node that adds several inputs stores its result in the precision of its first input. In float64, against the reference with its weights rounded to bf16 the same way, the extended and strict DeiT-Tiny graphs and the Hadamard graph with exact GELU, all with all keys, reproduce the reference logits on 16 Pet test images to 6×10^{-8}. The differences between the compiled and reference models in Table 2 therefore come from bf16 weights and arithmetic. The implementation of the strict graph evaluates $(x + \varepsilon)^{-1/2}$ as a single kernel; it equals the inverse square root after a fixed affine map.

Code availability

The compiler, the experiment configurations and the scripts that produce every table and figure are released under the MIT license at `https://github.com/iceslice0/transformer-to-dnn`.

Data availability

All datasets are public: Oxford-IIIT Pet [28], ImageNet-1k [29], DIV2K [58], Set5 [57] and WikiText-2 [59]. Pretrained weights come from timm, the MambaIRv2 release and the Hugging Face hub. The additional numerical results are reported in the tables and figures of the Supplementary Information.

References

[1] Vaswani, A. *et al.* Attention is all you need. In *Advances in Neural Information Processing Systems*, vol. 30, 5998–6008 (2017). URL `https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html`.

[2] Google Coral. TensorFlow models on the edge TPU. `https://www.coral.ai/docs/edgetpu/models-intro/` (2026). Accessed 2026-05-05.

[3] NVIDIA. TensorRT documentation: DLA supported layers and restrictions. https://docs.nvidia.com/deeplearning/tensorrt/latest/inference-library/dla-layer-restrictions.html (2026). Accessed 2026-10-03.

[4] Xu, D. *et al.* Fast on-device LLM inference with NPUs. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (2025).

[5] Park, J., Yoon, H., Ahn, D., Choi, J. & Kim, J.-J. OPTIMUS: OPTImized matrix MUltiplication structure for transformer neural network accelerator. In *Proceedings of Machine Learning and Systems (MLSys)*, vol. 2 (2020). URL https://proceedings.mlsys.org/paper_files/paper/2020/hash/91ba7292e5388b90b58d0b839a7f19ec-Abstract.html.

[6] Seo, M. *et al.* IANUS: Integrated accelerator based on NPU-PIM unified memory system. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)* (2024).

[7] Kim, J., Kim, S., Choi, K. & Park, I.-C. Hardware-efficient SoftMax architecture with bit-wise exponentiation and reciprocal calculation. *IEEE Transactions on Circuits and Systems I: Regular Papers* **71**, 4574–4585 (2024).

[8] Liu, Z. *et al.* Post-training quantization for vision transformer. In *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 34 (2021). URL https://proceedings.neurips.cc/paper/2021/hash/ec8956637a99787bd197eacd77acce5e-Abstract.html.

[9] Yuan, Z., Xue, C., Chen, Y., Wu, Q. & Sun, G. PTQ4ViT: Post-training quantization for vision transformers with twin uniform quantization. In *Proceedings of the European Conference on Computer Vision (ECCV)* (2022).

[10] Lin, Y., Zhang, T., Sun, P., Li, Z. & Zhou, S. FQ-ViT: Post-training quantization for fully quantized vision transformer. In *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence (IJCAI)* (2022).

[11] Li, Z., Xiao, J., Yang, L. & Gu, Q. RepQ-ViT: Scale reparameterization for post-training quantization of vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)* (2023).

[12] Moon, J., Kim, D., Cheon, J. & Ham, B. Instance-aware group quantization for vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2024).

[13] Kim, S., Gholami, A., Yao, Z., Mahoney, M. W. & Keutzer, K. I-BERT: Integer-only BERT quantization. In *Proceedings of the 38th International Conference on Machine Learning*, vol. 139 of *Proceedings of Machine Learning Research*, 5506–5518 (PMLR, 2021). URL https://proceedings.mlr.press/v139/kim21d.html.

[14] Xiao, G. *et al.* SmoothQuant: Accurate and efficient post-training quantization for large language models. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, vol. 202 of *Proceedings of Machine Learning Research*, 38087–38099 (PMLR, 2023). URL https://proceedings.mlr.press/v202/xiao23c.html.

[15] ONNX. LayerNormalization (opset 17), function definition. https://onnx.ai/onnx/operators/onnx__LayerNormalization.html (2026). Accessed 2026-10-03.

[16] ONNX. Softmax (opset 13), function definition. https://onnx.ai/onnx/operators/onnx__Softmax.html (2026). Accessed 2026-10-03.

[17] AMD. Vitis AI ONNX operators for Ryzen AI NPUs. https://vitisai.docs.amd.com/projects/gen2/en/latest/docs/additional_information/operators.html (2026). Accessed 2026-10-03.

[18] Arm. Ethos-U Vela compiler 5.2.0: Supported operators. https://gitlab.arm.com/artificial-intelligence/ethos-u/ethos-u-vela/-/blob/main/SUPPORTED_OPS.md (2026). Accessed 2026-10-03.

[19] Hu, J., Shen, L. & Sun, G. Squeeze-and-excitation networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 7132–7141 (2018).

[20] Howard, A. *et al.* Searching for MobileNetV3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 1314–1324 (2019).

[21] Gupta, A., Dar, G., Goodman, S., Ciprut, D. & Berant, J. Memory-efficient transformers via top-k attention. In *Proceedings of the Second Workshop on Simple and Efficient Natural Language Processing*, 39–52 (Association for Computational Linguistics, 2021). URL https://aclanthology.org/2021.sustainlp-1.5/.

[22] Wang, H., Zhang, Z. & Han, S. SpAtten: Efficient sparse attention architecture with cascade token and head pruning. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 97–110 (2021).

[23] Gong, P. *et al.* HATA: Trainable and hardware-efficient hash-aware top-k attention for scalable large model inference. In *Findings of the Association for Computational Linguistics: ACL 2025*, 24856–24871 (2025). URL https://aclanthology.org/2025.findings-acl.1275/.

[24] Geshkovski, B., Letrouit, C., Polyanskiy, Y. & Rigollet, P. A mathematical perspective on Transformers. *Bulletin of the American Mathematical Society* **62**, 427–479 (2025). URL https://arxiv.org/abs/2312.10794.

[25] Vyas, A., Katharopoulos, A. & Fleuret, F. Fast transformers with clustered attention. In *Advances in Neural Information Processing Systems*, vol. 33 (2020). URL https://proceedings.neurips.cc/paper_files/paper/2020/file/f6a8dd1c954c8506aadc764cc32b895e-Paper.pdf.

[26] Ballerini, M. *et al.* Interaction ruling animal collective behavior depends on topological rather than metric distance: Evidence from a field study. *Proceedings of the National Academy of Sciences* **105**, 1232–1237 (2008).

[27] Touvron, H. *et al.* Training data-efficient image transformers & distillation through attention. In *Proceedings of the 38th International Conference on Machine Learning*, vol. 139 of *Proceedings of Machine Learning Research*, 10347–10357 (PMLR, 2021). URL https://proceedings.mlr.press/v139/touvron21a.html.

[28] Parkhi, O. M., Vedaldi, A., Zisserman, A. & Jawahar, C. V. Cats and dogs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 3498–3505 (2012).

[29] Deng, J. *et al.* ImageNet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 248–255 (2009).

[30] Guo, H. *et al.* MambaIRv2: Attentive state space restoration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 28124–28133 (2025).

[31] Biderman, S. *et al.* Pythia: A suite for analyzing large language models across training and scaling. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, vol. 202 of *Proceedings of Machine Learning Research*, 2397–2430 (PMLR, 2023). URL `https://proceedings.mlr.press/v202/biderman23a.html`.

[32] Cybenko, G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems* **2**, 303–314 (1989).

[33] Pinkus, A. Approximation theory of the MLP model in neural networks. *Acta Numerica* **8**, 143–195 (1999).

[34] Laundy, S. L. *Table of Quarter-Squares of All Integer Numbers, up to 100,000, by which the Product of Two Factors May Be Found by the Aid of Addition and Subtraction Alone* (Charles and Edwin Layton, London, 1856).

[35] Winograd, S. A new algorithm for inner product. *IEEE Transactions on Computers* **C-17**, 693–694 (1968).

[36] Yarotsky, D. Error bounds for approximations with deep ReLU networks. *Neural Networks* **94**, 103–114 (2017).

[37] Rotem, N. *et al.* Glow: Graph lowering compiler techniques for neural networks. arXiv:1805.00907 (2018).

[38] Chen, T. *et al.* THE-X: Privacy-preserving transformer inference with homomorphic encryption. In *Findings of the Association for Computational Linguistics: ACL 2022* (2022).

[39] Hao, M. *et al.* Iron: Private inference on transformers. In *Advances in Neural Information Processing Systems*, vol. 35 (2022).

[40] Pang, Q., Zhu, J., Möllering, H., Zheng, W. & Schneider, T. BOLT: Privacy-preserving, accurate and efficient inference for transformers. In *IEEE Symposium on Security and Privacy (S&P)* (2024).

[41] Kosson, A. & Jaggi, M. Multiplication-free transformer training via piecewise affine operations. In *Advances in Neural Information Processing Systems*, vol. 36 (2023).

[42] Zhang, B. & Sennrich, R. Root mean square layer normalization. In *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, 12360–12371 (2019). URL `https://proceedings.neurips.cc/paper/2019/hash/1e8a19426224ca89e83cef47f1e7f53b-Abstract.html`.

[43] Bachlechner, T., Majumder, B. P., Mao, H., Cottrell, G. & McAuley, J. ReZero is all you need: Fast convergence at large depth. In *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*, vol. 161 of *Proceedings of Machine Learning Research*, 1352–1361 (PMLR, 2021). URL https://proceedings.mlr.press/v161/bachlechner21a.html.

[44] Zhu, J., Chen, X., He, K., LeCun, Y. & Liu, Z. Transformers without normalization. *arXiv preprint arXiv:2503.10622* (2025).

[45] Chen, B. *et al.* Scatterbrain: Unifying sparse and low-rank attention approximation. In *Advances in Neural Information Processing Systems*, vol. 34 (2021). URL https://proceedings.neurips.cc/paper/2021/hash/9185f3ec501c674c7c788464a36e7fb3-Abstract.html.

[46] Ren, H. *et al.* Combiner: Full attention transformer with sparse computation cost. In *Advances in Neural Information Processing Systems*, vol. 34 (2021). URL https://proceedings.neurips.cc/paper/2021/hash/bd4a6d0563e0604510989eb8f9ff71f5-Abstract.html.

[47] Chen, Z. *et al.* MagicPIG: LSH sampling for efficient LLM generation. In *International Conference on Learning Representations (ICLR)* (2025). URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/6d50d824ae819d5a961c1d8edc15e833-Abstract-Conference.html.

[48] Yu, J. *et al.* NN-LUT: Neural approximation of non-linear operations for efficient transformer inference. In *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC)*, 577–582 (2022).

[49] Upadhyay, M., Juneja, R., Wong, W.-F. & Peh, L.-S. NOVA: NoC-based vector unit for mapping attention layers on a CNN accelerator. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 1–6 (2024).

[50] Hendrycks, D. & Gimpel, K. Gaussian error linear units (GELUs). *arXiv preprint arXiv:1606.08415* (2016).

[51] Motsch, S. & Tadmor, E. A new model for self-organized dynamics and its flocking behavior. *Journal of Statistical Physics* **144**, 923–947 (2011).

[52] Kim, H., Papamakarios, G. & Mnih, A. The Lipschitz constant of self-attention. In *Proceedings of the 38th International Conference on Machine Learning*, vol. 139 of *Proceedings of Machine Learning Research*, 5562–5571 (PMLR, 2021). URL https://proceedings.mlr.press/v139/kim21i.html.

[53] Jacob, B. *et al.* Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2704–2713 (2018).

[54] Nagel, M. *et al.* A white paper on neural network quantization. arXiv:2106.08295 (2021).

[55] Hinton, G., Vinyals, O. & Dean, J. Distilling the knowledge in a neural network. arXiv:1503.02531 (2015).

[56] Jeffreys, H. An invariant form for the prior probability in estimation problems. *Proceedings of the Royal Society of London. Series A* **186**, 453–461 (1946).

[57] Bevilacqua, M., Roumy, A., Guillemot, C. & Alberi Morel, M.-L. Low-complexity single-image super-resolution based on nonnegative neighbor embedding. In *Proceedings of the British Machine Vision Conference (BMVC)*, 135.1–135.10 (BMVA Press, 2012). URL `https://bmva-archive.org.uk/bmvc/2012/BMVC/paper135/index.html`.

[58] Agustsson, E. & Timofte, R. NTIRE 2017 challenge on single image super-resolution: Dataset and study. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* (2017).

[59] Merity, S., Xiong, C., Bradbury, J. & Socher, R. Pointer sentinel mixture models. In *International Conference on Learning Representations (ICLR)* (2017). URL `https://openreview.net/forum?id=Byj72udxe`.

[60] Shazeer, N. GLU variants improve transformer. arXiv:2002.05202 (2020).

[61] Erofeeva, V., Granichin, O., Avros, R. & Volkovich, Z. Multilevel modeling and control of dynamic systems. *Scientific Reports* **14**, 27903 (2024).

[62] Wightman, R. PyTorch image models. `https://github.com/huggingface/pytorch-image-models` (2019).

[63] NVIDIA. NVIDIA Ada GPU architecture. Whitepaper, version 2.02, `https://images.nvidia.com/aem-dam/Solutions/Data-Center/l4/nvidia-ada-gpu-architecture-whitepaper-V2.02.pdf` (2023).

Funding

This work was supported by the Ministry of Economic Development of the Russian Federation (Agreement No. 139-15-2025-007, dated April 16, 2025; ID: 000000C313925P3O0002).

Author contributions

S.S. conceived the work, developed the theory, implemented the compiler, ran the experiments and wrote the manuscript. Z.V. provided support and supervised the work. O.G. proposed the clustering and multi-agent extension and acquired the funding. All authors reviewed the manuscript.

Competing interests

The authors declare no competing interests.

Supplementary Files

This is a list of supplementary files associated with this preprint. Click to download.

- supplementary.pdf